Available online at www.sciencedirect.com
ScienceDirect
Future Generation Computer Systems 23 (2007) 904–912
FGCS
FUTURE GENERATION COMPUTER SYSTEMS
www.elsevier.com/locate/fgcs

Sabotage-tolerance and trust management in desktop grid computing

Patricio Domingues[a], Bruno Sousa[b], Luis Moura Silva[b,*]

[a] School of Technology and Management, Polytechnic Institute of Leiria, Portugal
[b] Dep. Engenharia Informática, University of Coimbra, Polo II, 3030-Coimbra, Portugal

Received 19 April 2006; received in revised form 30 November 2006; accepted 5 December 2006
Available online 23 January 2007

Abstract

The success of grid computing in open environments like the Internet is highly dependent on the adoption of mechanisms to detect failures and malicious sabotage attempts. It is also required to maintain a trust management system that permits one to distinguish the trustable from the non-trustable participants in a global computation. Without these mechanisms, users with data-critical applications will never rely on desktop grids, and will rather prefer to support higher costs to run their computations in closed and secure computing systems.

This paper discusses the topics of sabotage-tolerance and trust management. After reviewing the state-of-the-art, we present two novel techniques: a mechanism for sabotage detection and a protocol for distributed trust management. The proposed techniques are targeted at the paradigm of volunteer-based computing commonly used on desktop grids.
© 2006 Elsevier B.V. All rights reserved.

Keywords: Desktop grids; Sabotage-tolerance; Trust management; Dependability

1. Introduction

In the past years, several initiatives of desktop grid computing have shown the potential opportunity for exploiting the idle CPU cycles that can be found in millions of Internet computers. Sound examples include SETI@home, Climateprediction.net, Einstein@home, among several others [1]. To support such global computations, there have been some notable advances in desktop grid middleware, with the emergence of open source platforms such as BOINC [2] and XtremWeb [3].

The verification of results is an important issue that needs to be addressed in any volunteer computation. Indeed, hardware and software mishaps as well as malicious volunteers can falsify the outcome of computations, rendering the results useless. Thus, a major concern of middleware tools supporting volunteer computation is to provide results validation and sabotage tolerance mechanisms. Since computations are run in open and non-trustable environments, it is necessary to protect the integrity of data and to validate the computation results. Without a sabotage detection mechanism, a malicious user can potentially undermine a computation that may have been executing for weeks or even months. Therefore, it is no surprise that users with computationally demanding applications do not easily trust open environments, rather preferring to have their applications executed over more controlled clusters which offer some reliability and trustability. This means that sabotage-tolerance is a mandatory issue in desktop grids in order to make them trustable and dependable. In this paper, we discuss the existing contributions and we present initial ideas for a new sabotage-tolerance mechanism targeted at real desktop grid initiatives.

Along with sabotage-tolerance techniques, it is crucial to devise protocols for trust management in desktop grids. For this purpose, low-level techniques are employed to gather valuable information for the creation and maintenance of local reputation lists. On top of that, higher level protocols are needed for globally sharing and maintaining an updated view of the participants' reputation. Some trust management systems have already been proposed in the area of Grid, like the Grid EigenTrust framework [4] and the EigenTrust system for P2P networks [5], among some other proposals [6]. However, these trust management systems do not properly exploit the computational paradigm of volunteer-based computing. In this paper, we propose an invitation-based protocol

* Corresponding author.
E-mail addresses: patricio@estg.ipleiria.pt (P. Domingues), bmsousa@dei.uc.pt (B. Sousa), luis@dei.uc.pt (L. Moura Silva).

0167-739X/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2006.12.001

for trust management targeted at volunteer desktop grids. The protocol establishes and updates the reputation of the participants according to their relationship in the volunteer-chain, using underlying sabotage-tolerance mechanisms to detect sabotage attempts to undermine the computations, or simply, computation errors due to faulty hardware.

The rest of this paper is organized as follows: Section 2 describes the state-of-the-art for sabotage-tolerance. Section 3 presents a novel mechanism to detect sabotage attempts, based on checkpoint comparison. Section 4 describes the state-of-the-art of trust management in distributed and P2P systems, while Section 5 outlines our protocol for a reputation system in desktop grids and explains some of its novelties. Finally, Section 6 concludes the paper and presents some insights about future work.

2. Sabotage-tolerance techniques

The master–worker model is the common paradigm for computing over desktop grids. Under this model, an application is broken into a large set of individual tasks, with tasks being distributed for computation by the master (also referred to as the *supervisor*) to request workers. After having processed a task, a worker sends the computed results to the supervisor. In an open environment like the Internet, it is necessary to assess the integrity and correctness of the results, since any host can run a worker.

The taxonomy of the sabotage-tolerance techniques can be classified in three distinct groups: (a) replication and voting; (b) sampling; and (c) checkpoint-based techniques. Next, we review each of these groups.

2.1. Replication and voting

The replication technique is also known as double-check [7] or as majority voting [8]. It was first deployed on a wide-scale by the SETI@home project to cope with erroneous results provoked by faulty hardware and malicious users eager to claim credits for work not performed [2]. The technique is based on the replication of individual tasks to different and preferably non-related workers. When completed, the results of the N replicas are compared and a majority voting is applied. The results that do not agree with the majority are marked as erroneous. If no majority can be determined (e.g. all results disagree), results are classified as erroneous and the task needs to be re-executed. N corresponds to the replication factor, and should be at least equal to two. The error rate of the replication method is determined by the replication factor N and by the percentage of erroneous/malicious volunteers. High levels of redundancy augment the resiliency at the cost of higher impact in the overall performance. For instance, the Einstein@home [9] project diminished its replication factor from 3 to 2 when it switched to a more computational demanding stage (S5), an evidence that replication can significantly consume computing resources. The main benefits of the replication approach are its support for generic computation and its simplicity, which eases its implementation — the technique is supported by the main desktop grid middlewares, and employed by all major public computing projects. On the contrary, a major weakness lies in the wasting of resources, since to complete a task, at least N instances need to be effectively computed. Furthermore, in computations that produce results sensible to hardware and software specificities, some further restrictions might be needed to support replication. For instance, some applications are extremely susceptible to floating-point implementations, and the same task run over different machines can yield different numerical results. A viable workaround is *homogeneous redundancy*, upon which replicas of a task are only assigned to homogeneous systems [11]. Regarding sabotage, the replication technique can be bypassed by smart colluding saboteurs as long as they manage to control a majority of replicas of a task. A more subtle limitation of replication-based validation for public computing environments is the potentially long interval that might elapse between the completion of the first result and the existence of enough results for majority voting. This is relevant in credit-based projects, where the effort of volunteers is rewarded through virtual credits. Indeed, credit assignment for a given task is only performed after the result has been validated, that is, after a majority of results matched and a so called *canonical result* exists. This means that the worker of the first result might wait a significant amount of time for receiving its due credits. Although this might be perceived as an irrelevant issue, credits and the associated tops, where users are ranked according to their earned credits, are major motivation factors for volunteers to participate in projects and thus everything related to credits should be treated carefully to avoid disgruntled volunteers [23].

2.2. Sampling techniques

Sampling techniques were developed to overcome the limitations of replication, namely its inefficient usage of resources. Sampling techniques are proposed under four different approaches: (a) naïve; (b) quizzes; (c) spot checks with black lists; and (d) ringers.

(a) *Naïve*. The naïve sample is a simple technique which uses probes to test the trustworthiness of participants [7]. Basically, the supervisor sends some test samples to the participants and then checks the results sent back by the assessed workers. However, the technique can be easily compromised by malicious workers if they are able to distinguish test samples from real application tasks. Indeed, a malicious worker can compute correctly the test samples, only faking application tasks, with its dual behavior possibly going unnoticed. The fact that test samples are computationally less demanding than real tasks makes the identification of test samples relatively easy and thus seriously compromises the usefulness of the technique. Furthermore, if the test samples are sent separately from the batch of real tasks, the detection of samples is even easier and the technique becomes almost useless in a hostile environment, as occurred in early versions of SETI@home [2].

Du et al. [7] extend the naïve sample technique by proposing the *commitment-based sampling* (CBS) approach for strictly one-way functions $f(x)$. Their goal is to hide the test samples, making them indistinguishable from real application tasks. CBS requires that a host, which computes $f(x)$ in the domain of D, saves all the intermediate results of its computation and builds a Merkle tree to prove that it effectively computed every input x. A Merkle tree is a hash-indexed binary tree, where data is kept on leafs and sibling nodes are built through a hash function. The CBS method involves the following four steps: (1) a participant computes its assigned tasks, locally building a Merkle tree which holds the intermediate results of the computation; (2) the supervisor sends a set of selected samples to the participant; (3) the participant proves its honesty by returning, along with the computed results, the Merkle tree's path up to the leaf node; (4) the supervisor verifies the results to check whether the participant is cheating or not. For that purpose, the supervisor reconstructs the Merkle tree. If the hash root node differs from the one reported by the participant, the participant is labeled as a cheater.

The main drawbacks of the CBS method are its limited applicability to one-way functions and the requirement that every worker builds and holds a possibly huge Merkle tree. Additionally, it induces a severe computational overhead on the supervisor due to the reconstruction of the Merkle tree.

(b) *Quizzes*. Shanyu and Lo [6] further extend the naïve sample method by hardening the detection of samples (labeled as *quizzes* by the authors). For that purpose, quizzes are mixed along with tasks. When a batch of tasks is finished, the supervisor checks the results related to the quizzes, only accepting the results if all quizzes are correct. Otherwise, the results are discarded and the tasks rescheduled for another execution. This method is resilient to collusion and presents the advantage that the outcome of samples can be verified before the end of a task [12]. However, no efficient method exists for generating quizzes in an automatic way, therefore preventing the use of this technique in wide-scale projects.

(c) *Spot checks with blacklists*. Spot-checking was proposed by Sarmenta [8]. This technique works similarly to quizzes. The main novelty is the tight integration of the technique with blacklists, which helps to filter out malicious users over time. When a participant is caught cheating, all her contributions until then are invalidated, and the participant is blacklisted and will be left out of any further computations. The implementation of spot-checking with blacklists faces some subtle problems, mainly the requirement of uniquely identifying participants over time. In fact, identification through email addresses, as it is commonly used by most volunteer projects is unreliable, since a malicious participant can easily and quickly obtain new email addresses.

(d) *Ringers*. Ringers were introduced by Golle and Mironov [13] to protect against coalitions of lazy cheaters assuming that all computational tasks involve the inversion of a strictly one-way function, $f(x)$, for a given value y. An example of the applicability of one-way functions is the attempt to break cryptographic functions through a brute-force approach, as is undertaken by the Distributed.net project [14]. Under the ringer approach, the supervisor creates individual tasks, each one involving a part D_i of the whole domain D. Before assigning a task, the supervisor adds to D_i a set of test samples (*ringers*) y_i, which are inverted values of D, computed through $y_i = f(x_i)$. Each task is then assigned to a worker w_i, which computes $f(x)$ for all x in its sub-domain D_i. A ringer y_i yields x_i, since $f(f(x_i)) = x_i$. Thus, to check the integrity of results, the supervisor just have to assess the x_i, which should correspond to the sent ringers y_i.

Two ringer-based versions have been proposed: *basic* and *bogus*. In the basic approach, when the supervisor assigns work to the participants, it includes a list of input values, for which it already knows the outcome, to be computed along with ringers. Each participant must then return the results yielded by the computation of input values and ringers, receiving credit only if all the ringers are effectively committed to the supervisor. A feebleness of this method is that the number of ringers is known by the participant. Therefore, a malicious participant can halt computation and return faked results as soon as all ringers of a task have been found. The bogus ringer version surmounts the limitations of the basic version by concealing the real number of ringers from the worker. For this purpose, a randomly chosen number of ringers whose results are of no interest ("bogus") are inserted in the computation set.

Szajda and Owen [15] tried to extend the ringers technique to generic computations, overcoming the one-way function limitation. In their approach, the supervisor plants ringers on the domain of values to be checked, with participants computing the values in the domain and the inserted ringers. However, the proposed approach is hardly feasible due to the huge difficulty of generating a method to automatically create the indistinguishable ringers.

2.3. Checkpoint-based verification

Monrose et al. [16] and Antonelli et al. [17], respectively, propose the (a) *basic checkpoint verification* and the (b) *distributed checkpoint verification*. Both schemes are checkpoint-based techniques for sabotage-tolerance and address sequential computations that can be broken into multiple temporal segments $(S_{t_1}, \ldots, S_{t_i}, \ldots, S_{t_n})$. At the end of each segment, a checkpoint $C(S_{t_i})$ of the task can be committed to stable storage. Next, we briefly review both techniques.

(a) *Basic checkpoint verification*. Under this technique, each worker periodically saves the state of its task in a checkpoint, computes its hash code and submits it to the supervisor. The supervisor randomly chooses a checkpoint-time S_{t_i} and requests the corresponding checkpoint $C(S_{t_i})$ from the worker. Then, the supervisor computes the partial execution of the task, from S_{t_i} up to the next checkpoint $C(S_{t_{i+1}})$. Finally, the hash code of $C(S_{t_{i+1}})$, that is, $H(C(S_{t_{i+1}}))$, is compared with the corresponding hash code sent by the worker.

The error rate of the basic checkpoint method depends on the number of checkpoints verified by the supervisor: a high percentage of verified checkpoints yields a low error rate at the cost of increased computation (for the partial computation of the task) and bandwidth (for having the checkpoint S_{t_i} transfered from the worker to the supervisor). Since, all of this overhead (computation and bandwidth) needs to be supported by the supervisor, this technique might induce an unbearable overhead to the supervisor, especially in wide-scale systems.

(b) *Distributed checkpoint verification*. Antonelli et al. [17] extended the basic verification technique by distributing the partial computation over workers. Their approach is comprised of six steps. Firstly, (1) the supervisor sends a task to the participant. (2) The worker then computes the results along with a list of the partial checkpoint hashes, sending both to the supervisor. (3) The supervisor stores the received hash list and selects a worker (henceforth the *verifier*) to verify it. The supervisor identifies the partial execution to be computed by the verifier and sends to the verifier the necessary data, namely how to contact the worker being scrutinized, so that it can obtain the checkpoint to load for the partial execution. (4) The verifier requests the initial checkpoint from the original participant, and then it (5) computes the partial task up to the next checkpoint, taking a hash code of this new checkpoint. Finally, (6) this hash code is sent to the supervisor which compares it with the one it received from the worker under assessment.

The distributed checkpoint verification method allows the verifications without overloading the supervisor. The intermediate steps can also be checked, allowing for the detection of a malicious worker before the completion of a task. The price for this technique is the redundancy required for checkpoint comparison, the cost of communications and the capability of participants to communicate directly with each other, a requirement that can be difficult to achieve when connectivity of hosts is restricted by firewalls and network address translation (NAT) schemes. Even if both machines can contact with each other, promoting direct contact between worker and verifier might create opportunities for collusion.

3. Validation through comparison of checkpoints

In this section, we combine replication with checkpoint-based comparison to promote early detection and finer localization of errors in volunteer computations. Specifically, we propose the *compare replicated checkpoint hashes* technique, and complement it with *trickle messaging* to permit early detection of divergent computations. We target public computing projects, assuming that a N-level replication is used for results validation.

3.1. Comparing replicated checkpoint hashes

Under the *compare replicated checkpoint hashes* (CRCH) approach, a worker is requested to return, along with the results of its task, a selected set of hashes of the checkpoints saved along the computation. The list of checkpoints whose hashes are requested is defined at task creation time, so that redundant instances of a task share the same set of requested checkpoint hashes. When a majority of replicated executions are completed, and thus the supervisor holds enough results for meaningful comparisons, the hashes from equivalent checkpoints are compared to each other. If a divergence occurs, the execution point where the differences were detected is marked as suspicious. Comparatively to the result comparisons and partial executions, the CRCH technique allows for a finer detection level, since an erroneous computation can be detected right after the first divergent checkpoint. For deterministic errors this might speed up the debugging process, since the temporal location of the fault is known with some precision, permitting a faster reproduction of the error.

Relatively to the *basic checkpoint* and to the *distributed checkpoint* techniques, CRCH requires no extra communications since the lightweight hashes can be sent to the supervisor along with the results. Additionally, the traditional communication model is not disrupted, since no contact is required between workers, contrary to the distributed checkpoint verification technique. Selective checkpoint hashing is also much less demanding for the supervisor, since no task computation (partial or complete) needs to be performed by the supervisor.

Although the CRCH strategy allows for result verification with practically no overhead at the server-side, and permits a more precise location of error occurrence, it does not speed up the detection of incorrect computations, since error detection can only occur after, at least, two replicas of the task have terminated. A more proactive variant is to have workers returning available checkpoint hashes during the computation. Ideally, from a detection point-of-view, the worker should send to the supervisor a hash immediately after its computation. However, such an attitude would increase the number of messages and consequently stress the supervisor network, possibly disturbing the whole system performance. A more realistic approach is to use the so-called *trickle messages* [25] to send checkpoint digests to the supervisor. A trickle message is sent by a worker to the supervisor and provides some status information about the worker. The trickle notification mechanism is used by projects like climateprediction.net [10] which have lengthy tasks (weeks or months long). It permits workers to update their progression status and to claim pending credits. Although the trickle designation covers a BOINC specific characteristic, the importance of this feedback mechanism for projects with long running tasks renders it mandatory for any serious desktop grid middleware. Thus, an improvement to the CRCH basic technique is to take advantage of the trickle messages which are already sent by workers to report status, for sending the hashes of the selected checkpoints without additional communication costs. This way, an error can be spotted by the supervisor as soon as a majority of checkpoint digests is available for the considered execution point. Thus, upon detection of a divergent computation, corrective measures can immediately be triggered by the supervisor. For instance, an additional instance of the task can be scheduled to replace the faulty task. Additionally, the thought-to-be faulty worker can be marked as suspect and further probed to assess its computational honesty, or, if repeating a faulty behavior, can be backlisted altogether [24].

4. Reputation systems

The auction site *eBay* [18] is a live example of the importance of reputation systems to promote transactions among individuals that do not know each other [19].

Indeed, reputation systems are important because they collect, distribute and aggregate feedback about participant's behavior and help to decide whom to trust, implicitly encouraging trustworthy behaviors. Next, we summarily review reputation systems for peer-to-peer and grids.

4.1. *Reputation systems on P2P networks*

Different reputation systems have been proposed for P2P networks: (a) debit-credit reputation computation; (b) credit-only reputation computation; (c) TrustMe; and (d) EigenTrust.

Minaxi et al. [20] devised the concept of debit-credit reputation computation (DCRC) and credit-only reputation computation (CORC) systems. Both systems give credits to users that serve others. DCRC also debits users who act as clients. In a P2P file-sharing context, both DCRC and CORC apply credit for file uploads, while DCRC also applies debits for file downloads. Under DCRC, collusion is not compensatory, since the user is debited when downloading, but using multiple identities – one for uploads and another for downloads – can be rewarding. On the contrary, CORC is resilient to multiple identities but is vulnerable to collusion, since colluded users can promote multiple downloads from each other only to receive undue credits.

Aameek et al. [21] proposed TrustMe, a secure and anonymous underlying protocol for trust management in P2P. TrustMe builds reputation in a user-based approach, in which a peer gives a reputation rating to another peer based on all its interactions with that peer. Therefore, the trust value of a peer A comes from the aggregation of all ratings that other peers have about A. To preserve anonymity, TrustMe keeps the trust rating of each peer in a random peer Y, called the *Trust Holding Agent* (THA). THA replies to all queries involving trust values it holds. TrustMe works as follows: when a peer A wants to know the trust value of peer B, it broadcasts a trust query. THA replies with the sought trust value and then, peer A decides whether or not to interact with B. If peer A resolves to interact with peer B, it will report to THA its own perceived trust level of B at the end of the iteration.

Kamvar et al. [5] suggested EigenTrust for P2P networks in order to build the global reputation of a peer A. EigenTrust is based on the aggregation of the local trust values that each peer holds about peer A. The algorithm operates on different steps: firstly, it normalizes the local trust values which must lie between 0 and 1; secondly, it aggregates the local trust values, collecting the opinions of different peers; finally, it makes a probabilistic interpretation of the results in order to search for reputable peers. Only the local trust value of peers that are not trying to mislead the system will be taken into account. The time and computational effort required to aggregate all the trust values are serious drawbacks of the system.

4.2. *Reputation systems for grid computing*

Alunkal et al. [4] proposed the Grid EigenTrust algorithm based on EigenTrust for P2P networks. The grid version works as follows: firstly, the algorithm establishes, within an institution, the trust value for each entity, based on various contexts. Then, it refers to reliability, which represents the trust of each institution. The global trust of an entity is derived from the institutions' trust reliability and the trust level of the entity as perceived within the institution. Fig. 1 illustrates how Grid EigenTrust organizes the entities (participants) inside an institution, where several institutions may belong to a virtual organization (VOs).

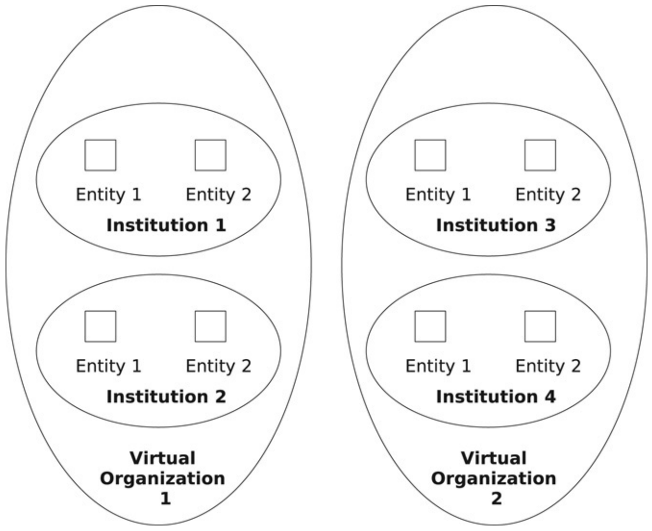

Fig. 1. The Grid EigenTrust reputation system.

The reputation services are responsible for evaluating the reputation of resources, services and users inside an institution and this evaluation is performed using the EigenTrust algorithm. The architecture of a reputation service is comprised of the following elements: *collection manager* (collects data from entities); *calculation manager* (computes reputation values based on a context); *data collection manager* (stores values to maintain a global and historical view) and the *reporter* (reports reputation values when queried). The Grid EigenTrust technique induces some overhead in communication and computation but it is suited for generic contexts, since it is not coupled to a specific application. It adapts dynamically to new participants/identities that join to a project inside an institution.

In their work, Shanyu and Lo [6] suggested a global reputation system where all participants share a trust list and an optional blacklist. The aggregation of the trust values reported by individual participants is made using simple functions. This technique does not incur computational overhead as does Grid EigenTrust, and adds the possibility to incorporate blacklists. Nevertheless, the study does not clearly explain how the simple functions can aggregate the trust values. Although some differences exist between the global reputation system and Grid EigenTrust, both share the same conceptual architecture.

5. Reputation through invitation

In this section, we propose the *volunteer invitation-based system* (VIS). This system aims at building trustable networks of volunteers resorting to invitations. Before presenting the volunteer invitation-based system, we discuss the problems related to uniquely identifying volunteer participants.

5.1. The identification problem

Uniquely identifying a volunteer participant is a serious challenge faced by trust management systems for volunteer desktop grid computing. Indeed, commonly used attributes like email addresses and host's IP addresses offer no guarantees of trustability and persistency. For instance, a malicious user can easily create an email account, in one of the many free email providers, for the sole purpose of engaging in a volunteer computing project in an anonymous way. If the malicious user behavior is caught by the sabotage tolerance system and the corresponding email address is blacklisted, the user can quickly create a new email account, and rejoin the project under a new and unsuspected identity. Likewise, IP addresses are not suited for unique and persistent identification of users, since most hosts are not directly connected to the Internet, instead being kept behind ISP or corporate firewalls and possibly with a masqueraded, private and dynamic IP address, that can vary periodically. Therefore, under such dynamic conditions, the IP address is meaningless for identification purposes. Furthermore, mobile computing devices like laptops allow their owners to easily connect from any geographical place they might be, further hardening a trustable identification through IP address.

Ironically, unique and reliable identification of users, if at all possible, also raises major privacy issues as the Pentium III's unique identifier number flaw demonstrated some years ago [22]. Moreover, unique identification schemes might discourage honest volunteers, not only from the burden identification schemes would probably require, but also for the loss of privacy they might represent to volunteers. After all, participants are volunteering their resources, and thus their effort in joining a project should be kept minimal, otherwise most potential volunteers will never participate. Therefore, in order to be usable, reputation systems should not depend on unique identification of users.

5.2. The volunteer invitation system

To circumvent the need for a unique and unforgeable identity system, we propose a novel approach, named the Volunteer Invitation-based System (VIS). VIS relies on human social relationships and on credit motivation to create a trustable and dedicated community of volunteers, where users invite other users to volunteer resources, vouching for their guests' trustability. In VIS, a user can only enroll as a volunteer in a desktop grid project through an invitation sent by a volunteer who is already contributing to the project. To insure that invitations are made in a conscious manner, inviters are rewarded or penalized according to the behavior of their guests. The goal is to make the inviters, up to a certain level, responsible for the behavior of the participants that have joined through their invitations. Under VIS, a volunteer participant who has proved her honesty and worthiness to the public-computing project, is granted a certain number of invitation cards. These invitation cards can be distributed to known users who want to join the volunteering network.

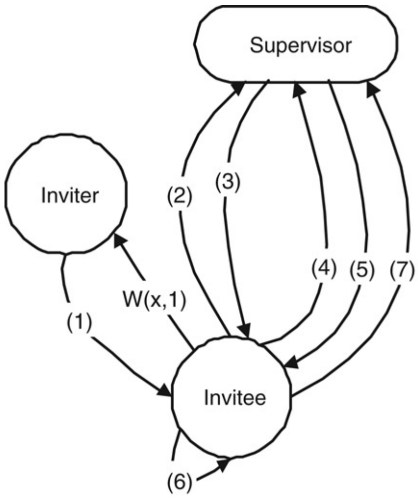

Fig. 2. Workflow of an invitation.

In VIS, the contribution of a participant is evaluated through the amount of donated work to the project (measured in credits), while honesty is assessed through sabotage tolerance measures. To motivate volunteers to recruit participants via their invitation cards, inviters receive a bonus given by a profit function $W(x, n)$, where x is related to the computing contribution achieved by participants that have enrolled through their invitations (n corresponds to the link depth between inviter and invitee, and is explained later on). Reciprocally, when an invitee is caught behaving in a dishonest manner, the inviter is penalized by the withdrawal of $L(x, n)$ amount of credits. The goal of the reward/penalty mechanism is to motivate volunteers to carefully choose the users they invite to join the volunteering network: a good invitation yields credits, while a badly chosen invitee provokes loss of credits. A basic outline of a new worker joining the volunteer project through an invitation is given in Fig. 2. Specifically, (1) the invitee receives the invitation, and (2) requests its activation to the project supervisor. Then, (3) the supervisor registers the new worker. Next, (4) a regular work cycle follows, with the worker requesting a task, (5) receiving it, (6) processing it, and (7) sending the results back to the supervisor. After having properly processed some tasks, the worker might receive some invitation cards, while the original inviter is awarded with $W(x, 1)$ bonus credits, with x corresponding to the credits directly earned by the invitee node.

Although the reliance of VIS on credit rewards and penalizations for motivating responsible behaviors might seem fragile, the importance of credit-based rewarding systems for public computing cannot be understated. In fact, although credits are merely virtual, and do not translate into any tangible asset, a significant number of volunteers donate their resources primarily for the thrill of earning credits and to move up on the project's rankings, regardless of the interest they might have for the problem(s) tackled by the public computing project [23]. In fact, to attest the importance of credits in the motivation of volunteers, it is rather common to have participants vigorously complaining about credit related issues in the user forums linked to the public projects.

Although existing public computing projects, which have an open enrollment policy (only a workable email address suffices for a volunteer to participate), can perform verification of results in accordance to a workers' reputation (with newcomer workers' results more frequently and thoroughly checked than long-time and supposedly honest volunteers), no mechanism exists in such projects to promote the recruitment of volunteers by current participants. On the contrary, by rewarding inviters with credits proportional to their invitees' performance, VIS stimulates active recruitment, and preferentially of good performers. Moreover, by penalizing inviters for ill-behaved invitees, VIS further fosters invitations of quality volunteers.

When a volunteer project is launched from scratch, the first invitations need to be sent by the project coordinators to credible users. Thereafter, a list of inviters will emerge, as a way to reward the most dedicated participants for their effort and commitment to the volunteer project. Over time, the chain of volunteers evolves, with participants that were once invited, receiving invitation cards to distribute and so on. An interesting open issue relates to the link strength, if any, that should exist between first generation inviters and non-directly invited descendants. As the name implies, a non-directly invited descendant is a participant that has received a invitation card through a former invitee of an inviter, and thus was not directly invited by the first generation participant. Formalizing, a nth-generation descendant is a user that was invited by a participant that was herself invited by a $(n-1)$th generation descendant and so forth. When n is one, we have a first generation descendant. Specifically, the open issue asks what should be the link between a participant and her descendants. That is, how much liability, if any, should an ascendant be held responsible for the acts of a nth generation descendant? Moreover, if held liable, how should this liability be accounted for? A possible solution would be to use a decay generation factor, upon which the impact of a descendant on the bonus and penalization credits, would be inversely proportional to generation distance between ascendant and descendant, possibly dropping to zero after a given number of generations. We plan to assess these issues in future work. Fig. 3 illustrates an inviter–invitee tree relationship, with $F(x, n)$ representing the credit payment yielded by a n-level invitee (due to space limitation not all inviter–invitee links are shown). Note that $F(x, n)$ corresponds to the artimethical sum of rewards $(W(x, n))$ and penalizations $(L(x, n))$.

An interesting feature of VIS for systems that resort to replication for error detection is the possibility to distribute redundant instances of tasks in a more informed manner. In fact, based on the inner knowledge of relationships between inviters and invitees, and to diminish the feasibility of collusion, the supervisor should distribute replicated instances of a task only to non-related workers. This way, the project avoids having related participants on the same voting group.

A possible misusage of the system would be for a participant who holds several machines to invite herself under a new identifier, trying to benefit from the credits that are given to inviters of well-behaved participants. For example, a user with three machines can in a first instance only register a machine, and when granted invitation cards, use them to enroll the other two machines under a newly created identifier. However, even if the sum of credits achieved by the multiple identifiers of the same user are superior to the credit granted to a single identifier with multiple machines, these credits are spread across multiple identifiers and might not be very fruitful in terms of ranking, except if identifiers are allowed to group as teams. Furthermore, self-invitation of a multiple-machine volunteer could be further discouraged by rewarding the volunteering of multiple machines in such a way that self-invitation would not yield additional earnings.

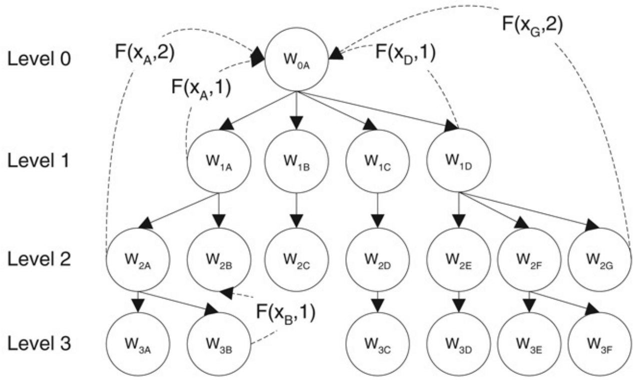

Fig. 3. Example of an invitation system relationship tree.

A potential limitation of the VIS system lies in the overhead that it might induce on the volunteer project supervisor. Indeed, inviter–invitee relationships need to be kept by the supervisor, and the relationship-tree might, especially in a wide-scale project, become unmanageable. Furthermore, dependent on the number of relationship generations kept and used for credit accounting, updating participants' credits might require expensive resources from the project server-side. However, relationship records can be limited to a certain generation-level and thus the induced overhead over the servers' project can be controlled at the cost of losing relationship related information.

5.3. Reputation across volunteer projects

The invitation-based system can be extended so that it supports recommendations of participants across multiple volunteer projects. The basic goal is to permit a volunteer who is already participating in a public project (or has participated in the past), to apply for an invitation in another project (from which the volunteer does not know anyone to ask directly for an invitation), presenting as references a virtual certificate provided by the project(s) she is currently participating in or has participated in in the past. This virtual certificate would include the worker performance and trustability metrics, such as the ratio of successful tasks completed, earned credits, and errors. Note that a certificate-based scheme could attenuate the possibly slow growth endured by a VIS-based system in its early stage, when the number of volunteers with invitation cards is still small.

The participation of a volunteer in multiple projects is not a novelty, and is actually promoted by the BOINC platform, which permits that a volunteer donates resources to several projects, specifying the CPU time distribution to be allocated to each project. The rationale for promoting multiple projects, which from the individual point of view of a project might seem counterproductive since the project loses exclusivity of resources, lies in the fact that many projects have downtime (for hardware and software maintenance and reparation of the server infrastructure), and shortage of tasks (for instance, when transitioning from one stage to another). Thus, participation in multiple projects helps to cope with a particular project downtime, besides permitting the volunteers to donate resources for several causes they might find worthy.

In terms of implementation, the virtual recommendation certificate could be a URL, unique to the participant/project pair, hosted by the project from which the participant is requesting references. The virtual certificate would be sent, on request, to the volunteer's registered email, and would have a limited time validity. Thus, when applying for an invitation to another project, the volunteer participant could attach its reference certificate(s) (the volunteer might already be participating in more than one project). Then, the project from which the volunteer is seeking an invitation could consult the reference certificate(s), analyze the metrics provided there, and decide accordingly whether it should or not deliver an invitation to the requesting volunteer.

The project-based reference certificate has the advantage of being simple, since it only requires a project to setup a secure web service capable of providing the participation metrics of a given participant. In fact, the BOINC framework already permits free web access to the work records of volunteer computers, which means that reference certificates should be straightforward to implement. A further benefit of the reference certificate would be to promote a volunteer using the same identification (email address) across all projects in which it already participates (or has participated in the past). To further stimulate adoption of unique identification across projects, a credit boost (or any other form of reward) could be assigned to a volunteer signing up with the same identification across projects.

6. Conclusion

This paper discussed two important topics in the area of grid computing: sabotage tolerance and trust management systems. We presented, to the best of our knowledge, new techniques which aim to exploit the specificities of a volunteer-based computing paradigm. Our work is still on-going and the next step will be the validation and enhancement of our techniques using a simulation framework, and then in a real system.

References

[1] Distributed computing, http://www.distributedcomputing.info/.
[2] D. Anderson, BOINC: A system for public-resource computing and storage, in: 5th IEEE/ACM International Workshop on Grid Computing, Pittsburgh, USA, 2004.
[3] C. Germain, V. Neri, G. Fedak, F. Cappello, Xtremweb: Building an experimental platform for global computing, in: First IEEE/ACM International Workshop on Grid Computing, Bangalore, India, 2000.
[4] B. Alunkal, I. Valjkovic, G. von Laszewski, K. Amin, Reputation-based grid resource selection, workshop on adaptive grid middleware (agridm 2003), in: Proceedings of the 12th International World Wide Web Conference, 2003.
[5] S. Kamvar, M. Schlosser, H. Garcia-Molina, The eigentrust algorithm for reputation management in P2P networks, in: Proceedings of the 12th International World Wide Web Conference, 2003.
[6] Z. Shanyu, V. Lo, Result verification and trust-based scheduling in open peer-to-peer cycle sharing systems, in: ICDS, 2005.
[7] W. Du, J. Jia, M. Mangal, M. Murugesan, Uncheatable grid computing, ICDS, IEEE Computer Science (2004).
[8] L. Sarmenta, Volunteer computing, Ph.D. Thesis, Department of Electrical Engineering and Computer Science, MIT, Cambridge, MA, December 2000.
[9] Einstein@home, http://www.einstein.phys.uwm.edu.
[10] climateprediction.net, http://www.climateprediction.net.
[11] M. Taufer, D. Anderson, P. Cicotti, C. Brooks III, Homogeneous redundancy: A technique to ensure integrity of molecular simulation results using public computing, in: 19th IEEE International Parallel and Distributed Processing Symposium, IPDPS'05, Workshop 1, April 2005.
[12] V. Lo, D. Zappala, D. Zhou, Y. Liu, S. Zhao, Cluster computing on the fly: P2P scheduling of idle cycles in the internet, in: IPTPS, 2004.
[13] P. Golle, I. Mironov, Uncheatable distributed computations, in: Proceedings of RSA Conference, Cryptographer's track, 2001.
[14] Distributed.net, http://www.distributed.net/.
[15] D. Szajda, B.G. Lawson, J. Owen, Hardening functions for large scale distributed computations, in: IEEE Symposium on Security and Privacy, IEEE Computer Science (2003).
[16] F. Monrose, P. Wyckoff, A. Rubin, Distributed execution with remote audit, 1999.
[17] A. Dominic, A. Cordero, A. Mettler, Securing distributed computation with untrusted participants, University of California at Berkeley, 2004.
[18] ebay auction site, http://www.ebay.com/.
[19] E. Friedman, K. Kuwabara, P. Resnick, R. Zeckhauser, Reputation systems: Facilitating trust on the internet, Communications of the ACM 43 (12) (2000).
[20] M. Gupta, P. Judge, M. Ammar, A reputation system for peer-to-peer networks, in: NOSSDAV, June 2003.
[21] A. Singh, L. Liu, Trustme: Anonymous management of trust relationships in decentralized p2p systems, in: IEEE International Conference on Peer-to-Peer Computing, ICP2PC, 2003.
[22] B. Schneier, Why Intel's id tracker won't work, http://www.news.zdnet.com/2100-9595_22-513519.html, 1999.
[23] A. Holohan, A. Garg, Collaboration online: The example of distributed computing, Journal of Computer-Mediated Communication 10 (4) (2005).
[24] L. Sarmenta, Sabotage-tolerance mechanisms for volunteer computing systems, in: 1st International Symposium on Cluster Computing and the Grid, 2001.
[25] C. Christensen, T. Aina, D. Stainforth, The challenge of volunteer computing with lengthy climate model simulations, in: 1st IEEE Conference on e-Science and Grid Computing, Melbourne, Australia, 2005.

Patricio Domingues received his B.Sc. and M.Sc. in Computer Engineering from the University of Coimbra, Portugal. He is currently a Ph.D. student at the University of Coimbra, and a teacher at the School of Technology and Management at the Polytechnic Institute of Leiria, also in Portugal. His main interests include scheduling and fault tolerance for desktop grids and peer-to-peer systems, beside script programming languages.

Bruno Sousa has a B.S. in Computer Engineering from the Polytechnic Institute of Leiria, Portugal. He is now a M.S. degree student at the Department of Computer Engineering, University of Coimbra. He can be reached at bmsousa@dei.uc.pt.

Luis Moura Silva is an Associate Professor at the Department of Engenharia Informatica from the University of Coimbra. He has a degree in Computer Engineering and a M.Sc. in Computer Science. His main research interests include: grid-computing, fault-tolerance, autonomic computing and distributed systems. He can be reached at luis@dei.uc.pt.